Flot de conception d'applications parallèles sur plateforme reconfigurable dynamiquement

Clément Foucher, Fabrice Muller, Alain Giulieri

RenPar'20 / **SympA'14** / CFSE 8
Saint-Malo, France, du 10 au 13 mai 2011

Flot de conception d'applications parallèles sur plateforme reconfigurable dynamiquement

Clément Foucher, Fabrice Muller et Alain Giulieri

Université de Nice-Sophia Antipolis
Laboratoire d'Électronique, Antennes et Télécommunications (LEAT)
CNRS UMR 6071, Bat. 4, 250 rue Albert Einstein
06560 Valbonne, France
{Clement.Foucher, Fabrice.Muller, Alain.Giulieri}@unice.fr

Résumé
Les architectures de calcul parallèle commencent progressivement à intégrer des unités matériellement reconfigurables. Ces unités peuvent accueillir différents circuits logiques, et donc implémenter des accélérateurs matériels en lien direct avec l'application exécutée. Notre objectif est de développer une nouvelle approche permettant une plus grande mixité entre les éléments de calcul logiciels et matériels afin de tirer parti de toute la puissance que peut apporter un accélérateur matériel pour l'exécution d'une tâche spécifique. À cette fin, nous proposons un flot destiné à la conception d'applications parallèles tout en gardant une compatibilité avec les applications déjà existantes. Nous présentons également une plateforme d'exécution destinée à ces applications et utilisant la reconfiguration dynamique partielle, qui permet de réduire la granularité de la reconfiguration, trop souvent limitée au FPGA entier. Dans ce papier, nous présentons le flot de conception ainsi que plusieurs implémentations de la plateforme.

Mots-clés : Architectures parallèles ; Systèmes reconfigurables ; System-on-chip ; FPGA

1. Introduction

Nous proposons dans ce papier un flot complet permettant l'implémentation d'une application parallèle à partir de ses spécifications en assemblant des accélérateurs matériels. Les applications cibles sont toutes les applications pouvant tirer parti à la fois d'une conception parallèle et d'accélérateurs matériels[1]. Se classent notamment dans cette catégorie les applications scientifiques destinées à être exécutées sur les superordinateurs, ainsi que les applications de traitement des flux de données.
Notre objectif est de réaliser ces applications de manière flexible afin de bénéficier d'un traitement auto-adaptatif en ligne prenant en compte la structure matérielle sous-jacente. Les applications sont ainsi scindées en sous-éléments indépendants pouvant être exécutés séparément, par exemple en utilisant des IPs dédiés. Ainsi, si l'on développe une application n'utilisant que des IP déjà existants, notre flot permet de se contenter de décrire les relations entre les blocs sans implémentation supplémentaire, permettant alors un temps de développement réellement court.
De plus, afin de mettre les unités d'exécution matérielles et logicielles sur un pied d'égalité, notamment dans le but de garantir une compatibilité avec le code logiciel déjà existant, nous considérons qu'un IP peut tout à fait être composé d'un processeur exécutant du code.
Dans cet objectif, nous avons également conçu une plateforme supportant ces applications. Cette plateforme doit être à la fois flexible pour permettre une utilisation sur différents types d'architectures cibles, et en même temps générique pour pouvoir accueillir différent types d'applications parallèles.
Dans ce papier, nous présentons le flot allant de la description de l'application jusqu'à la plateforme destinée à accueillir ces applications, basée sur des matrices reconfigurables de type FPGA (Field Programmable Gate Array – Matrice de portes programmables). Nous étudierons notamment la mise en place et les différentes implémentations de la plateforme parallèle, ainsi que ses futurs développements.

[1] Les accélérateurs matériels étant identifiés dans la littérature en tant que «Bloc de Propriété Intellectuelle» (Intellectual Property Block – IP Block), nous emploierons par la suite le terme d'IP.

La section 2 présente l'état de l'art en matière de parallélisme et de reconfiguration matérielle. La section 3 présente notre approche du monde parallèle et de la technologie reconfigurable, et les idées qui en ont découlé. La section 4 décrit notre flot de conception ainsi que les diverses implémentations de la plateforme que nous avons réalisée. Enfin, la section 5 conclut ce papier.

2. État de l'art

Cette section est séparée en deux parties : tout d'abord, l'état de l'art concernant le calcul parallèle, accompagné d'un bref historique, est présenté en section 2.1. Ensuite, nous nous intéressons particulièrement au parallélisme utilisant du matériel reconfigurable en section 2.2

2.1. Le calcul parallèle

La notion de calcul parallèle s'étend sur différentes échelles, depuis la simple puce contenant plusieurs cœurs de calcul jusqu'au superordinateur contenant plusieurs milliers de nœuds, chacun pourvu de plusieurs de ces puces.

2.1.1. Le parallélisme «on-chip»

Depuis quelques années, on assiste à un changement dans la manière d'augmenter les performances d'un processeur. En effet, après avoir principalement utilisé la montée en fréquence depuis les débuts de l'informatique, les fondeurs se tournent désormais vers l'augmentation du nombre de cœurs. La raison principale de ce changement est que la montée en fréquence est accompagnée d'une augmentation de la consommation énergétique et de la production de chaleur.

Les prémisses du parallélisme au sein d'une puce («on-chip») de type CPU (Central Processor Unit) remontent à la technologie Hyper-Threading d'Intel, qui permet d'exécuter virtuellement plusieurs threads simultanément. Pour cela, Intel duplique les registres du processeur tout en conservant les ressources d'exécution, permettant ainsi d'exécuter un thread lorsque l'autre est bloqué en attente. Les premiers processeurs réellement parallèles sont apparus en 2005, et continuent actuellement à se développer, le nombre de cœurs augmentant régulièrement.

Néanmoins, après plusieurs années de mise en œuvre du parallélisme, la plupart des programmes actuels peinent à utiliser le parallélisme qui leur est offert. En effet, leur conception reste, à de rares exceptions près, séquentielle. Heureusement, les systèmes d'exploitation modernes permettent l'exécution multitâche, ce qui autorise à utiliser différentes unités d'exécutions (Processing Elements – PEs) pour différents programmes tournant simultanément, et donc tirent néanmoins partie de ce parallélisme.

Les puces que nous avons mentionnées entrent dans la catégorie «multicore», ou multicœur. Mais une autre catégorie de parallélisme on-chip commence à se dessiner : le «manycore» («nombreux cœurs») [22, 21]. En effet, au delà d'une dizaine de PEs, le système actuel d'échange de données ne peut-être conservé sans dégrader fortement les performances. Dans ce cas, on utilise alors un paradigme équivalent à celui des réseaux, considérant chaque PE comme un nœud d'un réseau. Les PEs sont alors reliés entre eux par ce que l'on nomme un «Network On Chip» (NOC), ou réseau sur puce [4].

Au sein des puces multicore, le précurseur, du moins dans la catégorie commerciale, est le Cell Broadband Engine (Cell BE) d'IBM, Sony et Toshiba [14]. Processeur central utilisé dans la PlayStation 3, le Cell BE contient un cœur principal (PPE) de 64 bits, assisté par 8 cœurs secondaires (SPEs) de 128 bits. Les SPEs ont la particularité d'être de type Single Instruction Multiple Data (SIMD), ce qui leur permet de traiter simultanément 2 données de 64 bits, ou 4 de 32 bits, etc.

Concernant les puces de type manycore, il n'existe pas encore de puce commerciale à la connaissance des auteurs. Néanmoins, Intel propose une puce nommée Single-chip Cloud Computer (SCC) [15] à des fins de recherche. Le SCC dispose de 48 PEs de type pentium IA-32 reliés entre eux par un NOC.

Une autre achitecture que l'on peut qualifier de manycore est mppSoC [17], qui reproduit la topologie de calcul vectorielle des premiers HPCs. MppSoC utilise une architecture reposant sur un Array Control Unit (ACU), qui contrôle une matrice de PEs bidimensionnelle. Cette matrice permet de réaliser des opérations de type SIMD en distribuant une instruction à plusieurs PEs, qui l'exécuteront chacun sur des données différentes. La communication au sein de cette matrice repose sur deux réseaux complémentaires. D'un côté, MpNoC est un NOC permettant à n'importe quelle paire de PEs de communiquer. D'un autre côté, X-Net est un réseau local permettant à chaque PE de communiquer avec ses 8 voisins. Afin de gérer le parallélisme ainsi que la communication, le jeu d'instruction de l'ACU a été étendu pour

ajouter le contrôle de ce parallélisme.

2.1.2. Les superordinateurs

L'histoire des superordinateurs (High Performance Computers – HPCs) remonte à l'année 1976, lorsque Seymour Cray construit le Cray-1. Sans nous attarder sur l'historique, déjà réalisé par Erich Strohmaier [20], nous remarquons simplement que les premiers HPCs étaient bâtis sur des processeurs très particuliers, possédant une architecture vectorielle. Puis, avec le temps, les processeurs particuliers dédiés aux HPCs ont été remplacés par des processeurs grand public. Ainsi, les HPCs modernes sont bâtis comme des fermes de nœuds, chaque nœud étant composé d'un ou plusieurs processeur(s) comprenant chacun un ou plusieurs PE(s). A titre d'exemple, Jaguar, le HPC le plus puissant durant l'année 2010[2], est constitué de 37 376 processeurs AMD Opteron contenant chacun six cœurs de 64 bits.
La tendance actuelle est d'utiliser des GPGPUs (General Purpose Graphic Processor Unit) en complément des CPUs, tel Tianhe-1A, le nouveau numéro 1 au TOP 500, qui inclut des GPGPUs nVidia. En effet, les GPUs, initialement destinés au calcul graphique, proposent depuis longtemps un certain degré de parallélisme, très adapté au calcul graphique, et peuvent avantageusement remplacer un CPU pour certains types de calcul spécifiques. D'où l'extension aux GPGPUs, qui autorisent l'exécution de calculs plus généraux que les simples GPUs.
Un résultat intéressant à propos des HPCs est mentionné par Iushchenko [16]. En effet, il remarque à propos des travaux réalisés sur le classement TOP 500 (notamment ceux de Dongarra, l'un des créateurs du test LINPACK utilisé pour réaliser le classement TOP 500) qu'en moyenne, le premier ordinateur du classement se retrouve dernier après une période 6 à 8 ans, puis que les performances de celui-ci sont comparables aux ordinateurs grand public au bout d'une durée de 8 à 10 ans. Cela signifie que les performances d'un ordinateur premier au classement TOP 500 correspondent à celles d'un ordinateur grand public au bout d'une quinzaine d'années.
Sur le plan logiciel, les HPCs utilisent principalement le standard Message Passing Interface (MPI) [13], qui permet aux nœuds de communiquer entre eux par le biais de messages. Au sein des nœuds, OpenMP [6] permet une communication inter-PEs en utilisant la mémoire partagée.

2.2. Parallélisme et reconfiguration matérielle

La reconfiguration matérielle monte en puissance depuis quelques années, ouvrant de nouvelles portes quant à la puissance des applications. Son entrée récente dans le monde du parallélisme et des HPCs prouve l'intérêt que lui porte l'industrie.

2.2.1. Le matériel reconfigurable

La reconfiguration matérielle consiste à disposer d'un circuit logique générique permettant de réaliser n'importe quel type de circuit logique, dans la limite des ressources disponibles. Si la théorie existait depuis longtemps, la première implémentation réellement flexible vit le jour avec l'arrivée du FPGA en 1985, lorsque Xilinx proposa le XC2064. Initialement utilisés pour remplacer quelques dizaines de portes logiques, leur montée en puissance a permis à terme de réaliser des circuits complexes. Les FPGA modernes permettent par exemple d'intégrer plusieurs processeurs, ou divers IPs relativement complexes.
Si le FPGA remettait en cause la nature statique des circuits logiques, son handicap résidait dans son aspect monolithique : il ne pouvait être reprogrammé que d'un seul bloc. Cet impératif supprimait la possibilité de reconfigurer le système en cours d'utilisation, ou bien nécessitait d'adjoindre au FPGA un contrôleur externe capable de lancer la reconfiguration et gérer le système durant celle-ci.
L'arrivée récente de la Reconfiguration Dynamique Partielle (RDP), qui permet de modifier une zone du FPGA sans interrompre la partie complémentaire qui continue de fonctionner, a ouvert de nouveaux horizons pour les systèmes reconfigurables.

2.2.2. Architectures parallèles reconfigurables

En constante évolution, les HPCs ont commencé récemment à introduire des FPGAs en tant que PEs dans leurs architectures. Cet ajout permet de programmer matériellement un IP afin d'accélérer une section particulière du programme, en la réalisant à la place du CPU. De telles architectures sont alors appelées HP*R*C, pour High Performance Reconfigurable Computers.

[2] Selon le TOP 500 [1], qui classe les HPCs selon les résultats du benchmark parallèle High Performance Linpack (HPL).

Dans les architectures actuelles, les FPGAs sont vus comme des coprocesseurs. En effet, le programme principal continue d'être exécuté sur le processeur, celui-ci faisant appel au FPGA lorsqu'il détecte une portion du programme prévue pour être exécutée matériellement. A ce moment là, le CPU effectue une requête pour que lui soit attribué un FPGA (selon l'architecture, il peut également «posséder» un FPGA dédié), qu'il configure à l'aide du bitstream réalisant l'opération demandée.
Ainsi, dans les HPRCs actuels, les FPGAs sont contrôlés par les CPUs et reconfigurés en un seul bloc. Les premières expérimentations intégrant la reconfiguration partielle sur des HPRCs commerciaux, les Cray XD1, ont été menés par El-Araby et al. [9].
El-Ghazawi [10] distingue deux architectures de HPRC : le type Nonuniform Node, Uniform System (NNUS) et le type Uniform Node, Nonuniform System (UNNS). Dans les NNUS, chaque nœud du système possède à la fois des PEs logiciels (CPUs) et des PEs reconfigurables (FPGAs), et sont donc non uniformes, mais le système est uniforme car doté de ce seul type de nœud. A l'inverse, dans les UNNS, les nœuds sont uniformes, c'est à dire soit logiciel, soit reconfigurable, le système étant alors non uniforme car formé de ces deux types de nœuds.
Le projet européen *Æther* [2], qui impliquait certains auteurs de cet article, a présenté une architecture basée sur des éléments reconfigurables nommés Self-Adaptive Network Elements (SANEs) [5] montés en réseau. Chaque SANE est composé des cellules reconfigurables dynamiquement et communiquant entre elles. Chaque cellule pouvant voir son contenu adapté, les SANEs forment des entités adaptatives dont le comportement dépend des ces sous-éléments. Les SANEs sont ensuite montés en réseau pour former une application complexe reconfigurable et adaptative. Au sein du projet *Æther*, des travaux ont été réalisées sur l'ordonnancement des ressources sur les SANEs [18], ainsi que sur la simulation de l'environnement des SANEs et les stratégies d'ordonnancement [11].
Crenne [7] présente une méthodologie permettant la reconfiguration dynamique à partir de bitstreams provenant d'un serveur. Son approche évite un stockage local des bitstreams en procédant à une copie directe depuis les paquets vers l'ICAP (Internal Configuration Access Port) [23], qui est l'IP intégré par Xilinx au sein de ses FPGAs permettant de contrôler la reconfiguration partielle de l'intérieur du FPGA.

3. Motivations

Dans cette section, nous expliquons les motivations nous ayant mené à réaliser ces travaux de recherche dans la section 3.1, ainsi qu'une première approche de notre système dans la section 3.2.

3.1. Notre approche des applications dynamiques

L'architecture actuelle des structures d'exécution parallèles – et notamment des HPRCs –, avec le matériel reconfigurable vu comme une unité d'appoint pour épauler le logiciel souffre, à nos yeux, d'un héritage trop lourd. En effet, jusqu'à récemment, une plateforme ne pouvait exécuter des applications différentes qu'en ayant recours au logiciel. Et si parfois certains IPs permettent d'accélérer une partie de l'application, comme par exemple un décodeur MPEG2 sur une carte graphique de PC, il est impossible de disposer de tous les IP possibles et imaginables de manière statique. Logiquement, les architectures HPRC vont donc dans la continuité des architectures passées, en ajoutant un soupçon de matériel, mais sans remettre en cause le modèle actuel dans lequel le logiciel est aux commandes. Pourtant, et maintenant que le matériel n'est plus figé comme auparavant, nous commençons à disposer de moyens importants pour changer le rapport de force entre logiciel et matériel. En effet, aux yeux des auteurs, le matériel ne doit plus être vu comme une aide ponctuelle au logiciel, mais la totalité de l'application doit être conçue dans une logique de codesign logiciel–matériel en ligne. La notion de codesign en ligne permet de faire le choix d'une implémentation logicielle ou matérielle pour chaque élément d'une application, à la fois lors de sa conception, mais également dynamiquement en fonction des ressources disponibles, tant sur processeurs que sur zones reconfigurables. Par exemple, on choisira plutôt le logiciel pour toute la partie contrôle, comme l'ordonnancement, tandis que le matériel aura notre préférence pour la partie traitement des données et communications. On retrouve ainsi la partie étude qui prélude actuellement à la conception d'un ASIC (Application-Specific Integrated Circuit – Circuit intégré dédié à une application), à la différence que notre approche est flexible et non dédiée.

3.2. L'architecture proposée

Pour cette raison, nous proposons ici une architecture reconfigurable dans laquelle la distinction entre PEs logiciels et matériels n'a que peu d'importance, tous étant regroupés dans la notion de cellule. Les cellules communiquant entre elles pour réaliser une application complexe, celles-ci pourraient être comparées aux SANEs du projet Æther. La figure 1 présente une vue simple du système.

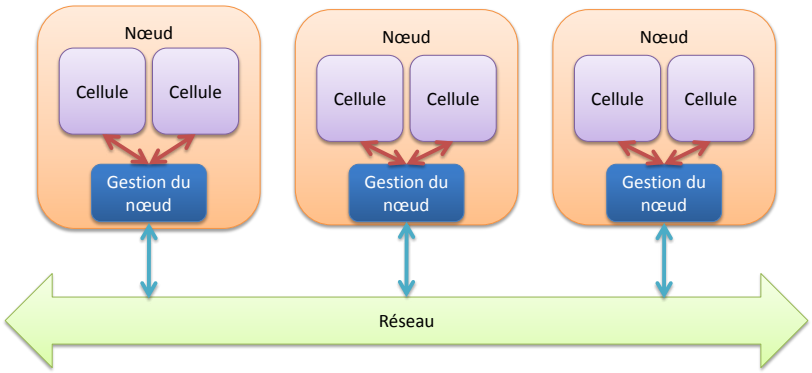

FIG. 1 – Vue du système.

Dans cette architecture, une application est découpée en diverses tâches séquentielles et/ou concurrentes. Chaque tâche peut être conçue logiciellement, matériellement, ou disposer des deux types d'implémentations. De la même manière, une tâche peut disposer de plusieurs implémentations matérielles utilisant des ressources différentes. Ainsi, une même tâche peut avoir une implémentation utilisant des DSP48Es, petits DSPs intégrés dans les FPGAs Xilinx, tandis qu'une autre pourra utiliser une zone du FPGA ne comportant pas de telles ressources, en remplaçant les DSP48Es par leur circuit logique équivalent au détriment des performances.
Notre objectif est de proposer une méthodologie de développement pouvant s'appliquer à la plupart des programmes parallèles, ainsi qu'une plateforme pouvant accueillir ces programmes, tout en conservant une compatibilité maximale avec des programmes déjà existants.
Notre architecture est distribuée, comme sur les HPCs actuels, sur différents nœuds, chacun contenant plusieurs cellules reconfigurables. Les cellules pouvant contenir indifféremment un PE logiciel ou matériel, notre architecture s'apparente au type NNUS détaillée précédemment.
Cette plateforme permet de «virtualiser», ou tout au moins de faire abstraction du matériel sous-jacent, en proposant un modèle unifié sur lequel les applications bâties en utilisant notre flot pourront s'exécuter. Les catégories d'applications pouvant tirer parti d'une telle plateforme sont celles pouvant nécessiter de l'auto-adaptation et/ou du parallélisme. Concernant le parallélisme, la plupart des applications scientifiques fonctionnant sur HPC gagneraient à déporter une partie de leurs calculs intensifs sur du matériel, comme le montre déjà la tendance HPRC. Notre approche permet d'aller plus loin en concevant l'application dès l'origine pour une architecture où le matériel est présent et flexible. Concernant l'auto-adaptativité, de nombreuses applications de traitement du signal en temps réel ont des besoins changeants en fonction du temps et doivent par exemple s'adapter aux conditions extérieures. Dans ce cas, une telle plateforme leur permettrait de réguler leurs besoins en puissance de calcul en fonction des PEs disponibles, et même de modifier le type de PE en fonction des besoins.

4. Notre système

Notre système se compose de deux éléments principaux : un flot de conception d'applications et un prototype de plateforme permettant d'accueillir ces applications. Dans cette section, nous présentons le flot en section 4.1, et nous abordons la plateforme associée pour ensuite discuter de ses évolutions en

section 4.2.

4.1. Flot de conception et modèle

Notre flot de conception, présenté sur la figure 2, débute par une description de l'application de type graphe. À cette fin, l'application est segmentée en noyaux applicatifs, un noyau étant un élément de l'application réalisant une tâche de calcul précise, par exemple une transformée de Fourier discrète. Pour une nouvelle application, le choix de la segmentation des noyaux peut être fait selon des critères de disponibilité des IPs. Par exemple, pour une application de transcodage de vidéo, on peut utiliser un IP de décodage du format d'entrée suivie d'un IP de codage vers le format de sortie. Dans le cas d'une application logicielle existante, on peut l'analyser en vue de la découper selon les différentes routines déjà existantes, pour ensuite en déplacer certaines vers des IPs matériels correspondant au même traitement.

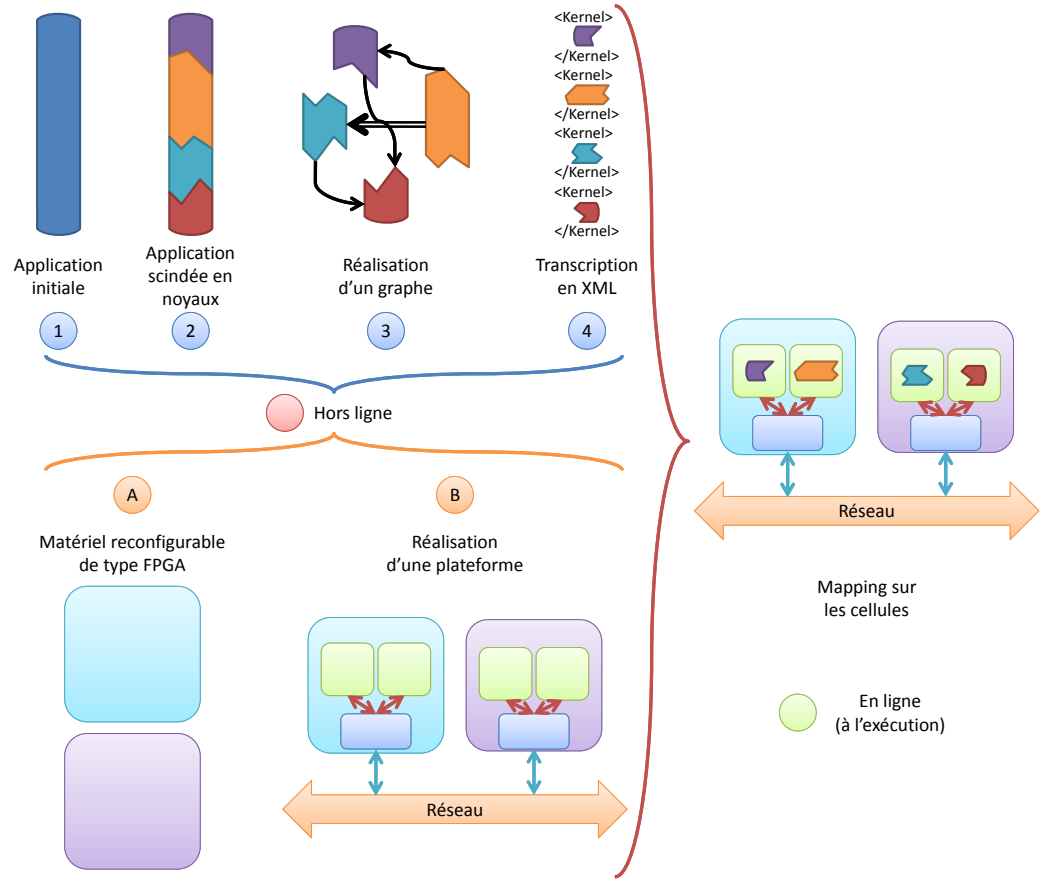

FIG. 2 – Flot de conception.

Une fois les noyaux isolés, on détermine les dépendances de données entre ces noyaux. On peut alors bâtir un graphe, indépendant du contenu des noyaux, permettant de visualiser les dépendances internes de l'application. Ceci permet d'identifier quels noyaux pourront être exécutés simultanément. Ce graphe est également utile pour identifier les échanges intensifs de données entre les noyaux, et ainsi détecter lesquels doivent bénéficier d'un placement minimisant le temps de communication entre eux. Le graphe étant une représentation abstraite de l'application, on se basera sur une description XML pour transformer celui-ci en un ensemble de données compréhensibles par un ordonnanceur.

TAB. 1 – Caractéristiques des différentes plateformes.

Nom de la plateforme	Dynamique	Communication via MPI	Type des IPs	Communications intra-nœud
Software HPC Platform	Non	Oui	Logiciels	Mémoire partagée
Hardware Stream Dynamic Platform	Oui	Non	Matériels	Mémoire partagée + bus
Hybrid HPRC& Stream Platform	Oui	Oui	Logiciels et Matériels	NOC

La définition du modèle utilisant les graphes est fonctionnelle, dans le sens où elle permet de modéliser plusieurs types d'applications et plusieurs modèles de parallélisme. Par exemple, elle permet de modéliser aussi bien des applications utilisant le standard MPI (Message Passing Interface) qu'OpenMP, bien que leur comportement diffère. En effet, MPI utilise un modèle dans lequel les threads existent du début à la fin d'une application, tandis qu'OpenMP utilise un modèle fork/join dans lequel les threads sont créés en fonction des besoins puis détruits lorsqu'ils deviennent inutiles.
Notre modèle utilise pour l'instant une représentation non standard, mais notre objectif est de le faire évoluer de manière à converger vers des graphes déjà existants, tels les automates finis parallèles [19]. Il devrait ainsi être possible de définir notre modèle en tant qu'extension d'un modèle déjà existant.

4.2. Notre plateforme et son évolution

Afin de développer notre plateforme, nous sommes passés par plusieurs étapes. Nos différentes plateformes prennent pour base l'architecture générale des HPCs, c'est à dire la division en nœuds contenant chacun plusieurs PEs, que nous appellerons ici cellules. La division en nœuds résulte en une architecture mémoire globalement distribuée – localement partagée. En effet, chaque nœud possède sa propre mémoire, celle-ci étant partagée entre les différentes cellules.
Afin de disposer d'une architecture reconfigurable, on implémente celle-ci sur une cible de type FPGA. Notre plateforme finale devra respecter les différents éléments déjà détaillés en section 3.2. A cette fin, nous détaillons la plateforme actuelle et les évolutions qui y seront apportées.
Une première plateforme, *Software HPC Platform*, a permis de valider le modèle général et son adéquation à la structure HPC, tout en pointant du doigt certains problèmes inhérents à la méthode d'utilisation de la mémoire. La deuxième plateforme, *Hardware Stream Dynamic Platform*, en cours de réalisation, devra permettre de prouver la viabilité de l'utilisation de la RDP dans une architecture orientée flot de données. Enfin, une future évolution, *Hybrid HPRC&Stream Platform*, devrait apporter une réponse complète en rassemblant les éléments des deux autres. Un résumé des caractéristiques des plateformes est présenté dans le tableau 1.

4.2.1. Software HPC Platform

Dans cette plateforme, les cellules de calcul sont toutes composées d'un PE logiciel statique basé sur un processeur de type MicroBlaze. En effet, cette plateforme préliminaire, présentée en [12], a servi à valider l'architecture globale de la plateforme. Une vue du système avec détail d'un nœud est présenté sur la figure 3. Afin de garantir une compatibilité avec les applications actuelles, cette plateforme appuie ses communications sur le standard industriel MPI via la librairie open-source MPICH2. Les communications inter-nœuds sont ainsi gérées par la cellule hôte, qui héberge un linux embarqué tournant sur un PowerPC 440. Les communications intra-nœud sont, elles, assurées par le biais de la mémoire partagée. Dans l'absolu, la communication via la mémoire partagée pourrait être gérée à l'aide d'OpenMP. Néanmoins, l'adaptation de cet outil à notre plateforme, dans laquelle les cellules de calcul sont dépourvues de système d'exploitation, aurait nécessité l'écriture d'un compilateur spécifique, ce qui ne rentre pas dans le cadre de cette simple plateforme de test. Ainsi, les deux types de communication sont gérées à l'aide de routines MPI.

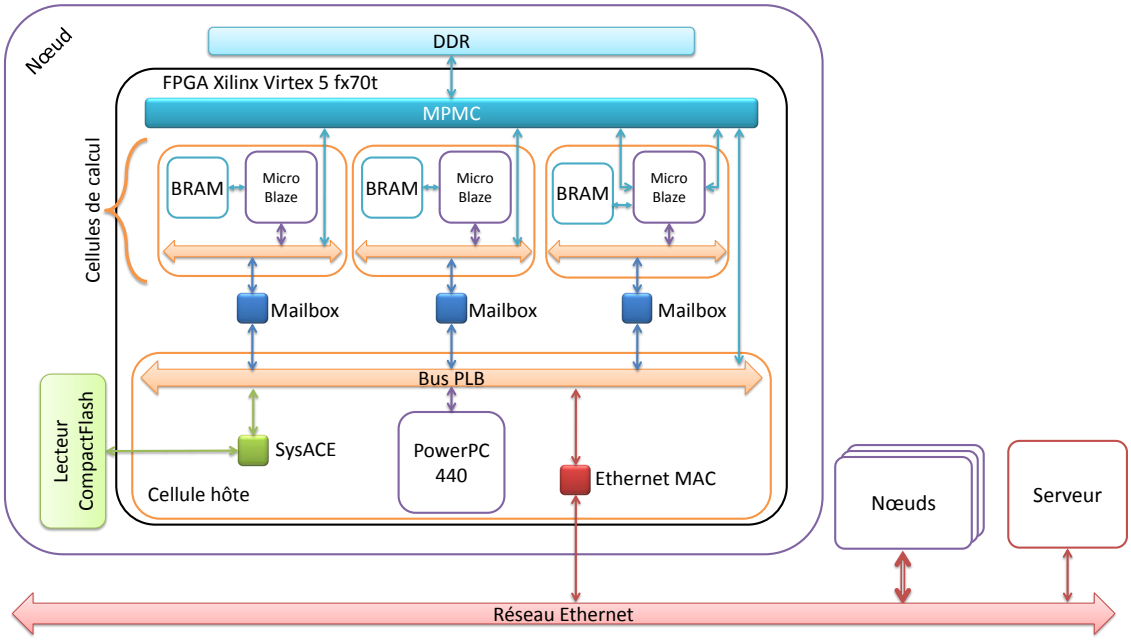

FIG. 3 – *Software HPC platform* avec détail sur un nœud.

Le système complet étant composé de plusieurs nœuds, il est nécessaire de disposer d'un point d'entrée pour lancer les calculs. Pour cela, on ajoute au système un nœud spécique, que l'on appelle serveur, et dont le rôle se résume à lancer les calculs sur les nœuds. Ce rôle peut être rempli par n'importe quelle machine disposant de MPICH2. Dans notre cas, il s'agit d'un simple poste de travail de type x86_64.
Cette première plateforme introduit des caractéristiques qui seront conservées par la suite, et notamment la cellule hôte qui permet de gérer les communications avec les autres nœuds.
Nous avons testé la plateforme à l'aide de l'application Integer Sort (IS) du benchmark NPB (NAS Parallel Benchmark) [3]. La plateforme est testée selon les critères suivants :
– la charge totale de calcul, représentée par la taille du bench, ou classe dans la terminologie NPB,
– le degré de parallélisation, représenté par le nombre total de cellules de calcul participant au bench,
– la charge locale de calcul, représentée par le nombre de cellules de calcul utilisées sur chaque nœud,
– la présence ou non de cache d'instruction dans les processeurs des cellules de calcul.
La figure 4 présente le temps d'exécution du bench selon diverses configurations, tandis que la figure 5 se focalise sur le temps passé par les différentes cellules à communiquer. Les résultats présentés sur les figures concernent des benchmarks de classe A. On constate sur les graphiques, et principalement sur celui présentant le temps de communication, qu'à autres paramètres égaux, le temps présenté en ordonnée croit lorsque l'on rassemble les cellules participant au bench sur les mêmes nœuds. En effet, la mémoire partagée est utilisée par toutes les cellules simultanément, ainsi que pour les communications intra-nœud, et se comporte comme un goulot d'étrangement, limitant sévèrement les performances. Ainsi, notre approche globale est confirmée par l'obtention une plateforme fonctionnelle malgré la mise en lumière de certains problèmes inhérents au mécanisme de communication intra-nœud.

4.2.2. Hardware Stream Dynamic Platform

Avec cette plateforme, nous nous concentrons sur l'aspect dynamique des IPs. A cet effet, cette architecture repose cette fois-ci sur des cellules matérielles, se basant sur des IPs existants. La figure 6 présente le système avec une configuration comportant deux nœuds, chacun pourvu de trois cellules de calcul. Elle ne dispose pas du standard MPI, qui n'est pas réellement nécessaire dans cette version dans la mesure où les IPs utilisés disposent de leur propre manière de récupérer et fournir des données, différente de MPI. Cette plateforme est donc moins orientée HPC, et plus flot de données. Elle permet l'exécution

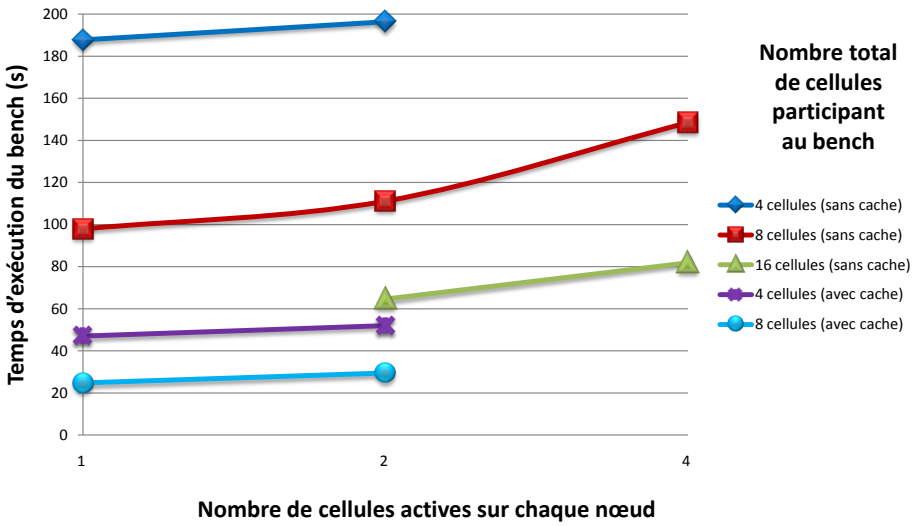

FIG. 4 – Résultats obtenus pour le temps total d'exécution du bench NPB IS avec différentes configurations de la plateforme.

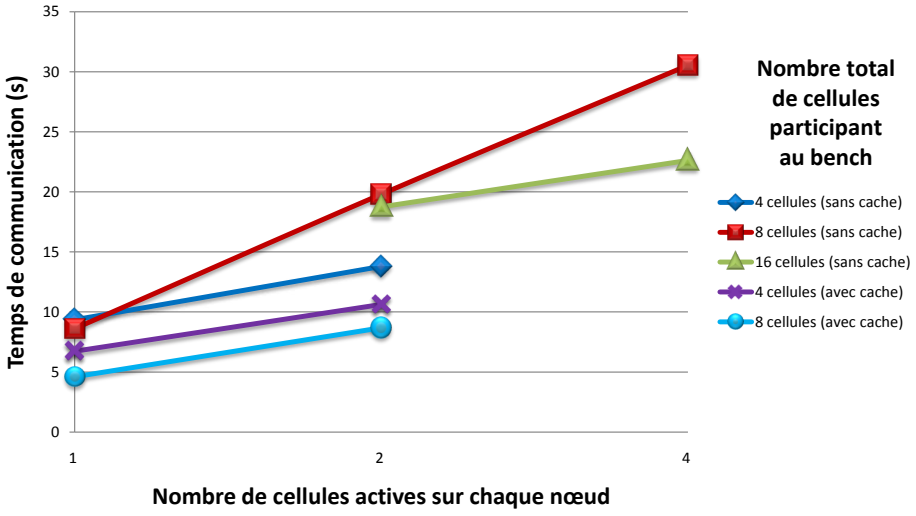

FIG. 5 – Résultats obtenus pour le temps de communication du bench NPB IS avec différentes configurations de la plateforme.

d'applications auto-adaptatives par la présence d'un ordonnanceur local répartissant dynamiquement les instances des noyaux, ou threads, sur les cellules.
Afin d'éviter de reproduire les problèmes de la précédente plateforme, les transferts de données peuvent cette fois emprunter le bus inter-cellules et ne reposent donc plus uniquement sur la mémoire partagée. Ici, le simple serveur de la plateforme précédente est remplacé par un ordonnanceur global, décidant de la répartition des tâches sur les nœuds. La nouveauté est que cet ordonnanceur global est secondé par

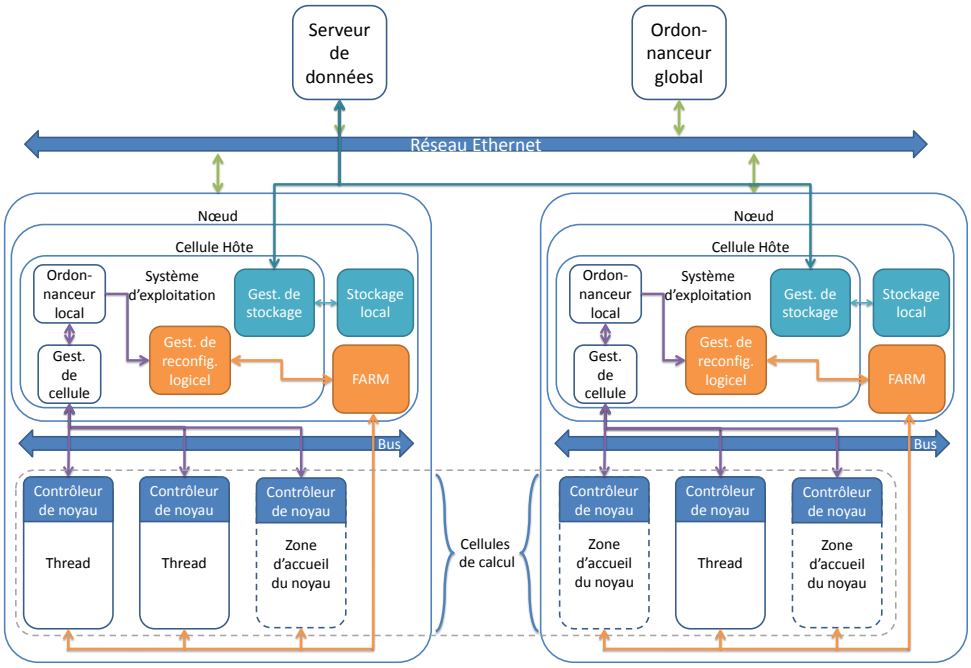

FIG. 6 – Vue de la *Hardware Stream Dynamic Platform* avec deux nœuds.

un ordonnanceur local, qui décide de la répartition des tâches au sein du nœud.
De plus, un serveur de données est ajouté, qui permet de distribuer les différents éléments de l'application, permettant ainsi un fonctionnement indépendant des nœuds. En effet, le serveur de données évite un stockage préalable des éléments de l'application sur chaque nœud, ces éléments étant récupérés dynamiquement sur le serveur de données.
Contrairement à la *Software HPC Platform*, dont toutes les cellules étaient homogènes, la *Hardware Stream Dynamic Platform* dispose de cellules dont les ressources peuvent différer. Ainsi, à la répartition figée de la plateforme statique succède une répartition dynamique prenant en compte les spécificités des cellules pour décider de la localisation d'un thread.
Concernant les cellules matérielles, nous devons gérer une dimension supplémentaire, qui se trouve être la reconfiguration partielle des cellules de calcul. Celle-ci s'appuie sur le gestionnaire de reconfiguration, qui épaule l'ordonnanceur local en s'occupant de réaliser les ordres de reconfigurations. Le gestionnaire de reconfiguration est composé de deux parties. La partie logicielle s'occupe de récupérer le bitstream partiel sur le serveur de fichiers, et de le stocker localement. Ainsi, à la différence de Crenne [7], nous stockons localement les bitstreams de manière à ce qu'ils soient déjà disponibles au moment où la reconfiguration doit s'effectuer. Afin de gérer l'encombrement local, les bitstreams sont remplacés sur une politique comparable au LRU (Least Recently Used) que l'on peut trouver dans les caches.
L'autre partie du gestionnaire de reconfiguration est matérielle et représente une surcouche de l'ICAP nommée Fast Reconfiguration Manager (FaRM) [8], qui a donc à sa disposition le bitstream et s'en sert pour configurer la cellule. Un driver Linux permet de commander l'IP FaRM à partir du gestionnaire de reconfiguration logiciel, transformant les commandes de reconfiguration en chargement de bitstreams partiels.
Cette plateforme, dont une implémentation est présentée sur la figure 7, est fonctionnelle du point de vue local, c'est à dire que chaque nœud peut fonctionner indépendamment si on lui communique la description de l'application à exécuter. L'ordonnanceur global aura pour tâche de découper une application en plusieurs éléments qui seront distribués sur les nœuds, avec pour objectif de minimiser les échanges de données inter-nœud, afin d'éviter les communications lentes.

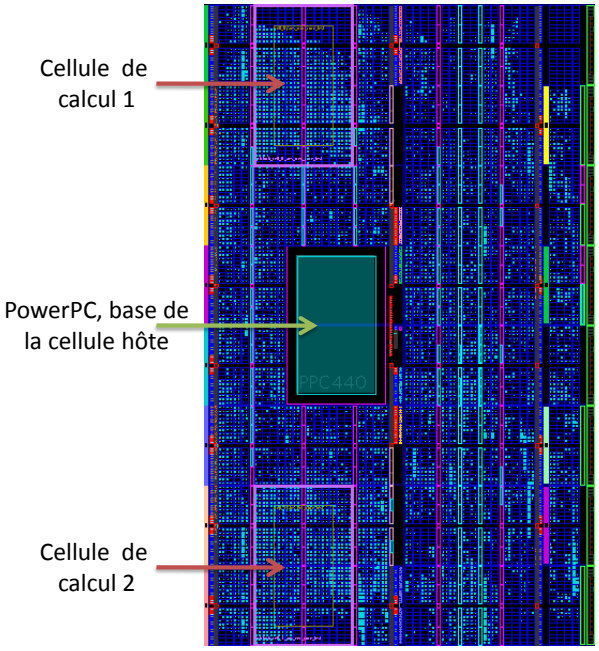

FIG. 7 – Implémentation d'un nœud de la *Hardware Stream Dynamic Platform* contenant deux cellules sur Virtex 5 FX70t.

4.2.3. Hybrid HPRC&Stream Platform

Les travaux futurs concerneront la mise en place d'une troisième plateforme reprenant et combinant les divers éléments des deux précédentes.
En tant que plateforme finale, il est prévu d'y intégrer la cohabitation entre les threads logiciels, constitués d'une routine, et les threads matériels, sous forme d'IP. Afin d'unifier la communication, on souhaite intégrer une gestion matérielle de MPI afin de donner accès à ce standard aux threads matériels de la même manière que les threads logiciels, et ainsi garantir la compatibilité d'anciennes applications en cas de portage sur notre plateforme.
Cette plateforme incluant à nouveau les cellules logicielles et la communication MPI, le même type de problèmes rencontrés avec la *Software HPC Platform* pourrait réapparaitre. Pour éviter cela, nous prévoyons d'implémenter la communication par le biais d'un NOC permettant la communication simultanée entre plusieurs couples maîtres–esclaves.

5. Conclusion

Dans ce papier, nous avons détaillé les différents éléments de notre méthodologie destinée à l'implémentation d'applications parallèles. Nous avons étudié le flot de conception de l'application, permettant nativement un codesign logiciel–matériel en ligne des applications. De plus, nous avons vu que ce flot pouvait facilement s'appliquer aux applications existantes désirant profiter de notre méthodologie.
Nous avons également abordé les différentes plateformes que nous avons développées pour permettre d'accueillir ces mêmes applications. Ces différentes plateformes ont mis en lumière les points faibles à résoudre dans les futures plateformes, tout en validant le concept de l'architecture générale.

Bibliographie

1. Site du classement TOP500 http ://top500.org.
2. Site web du projet Æther : www.aether-ist.org.
3. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frede-

rickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, et S. Weeratunga. *The NAS Parallel Benchmarks*. http ://www.nas.nasa.gov/News/Techreports/1994/PDF/RNR-94-007.pdf, 1994.
4. Luca Benini et Giovanni De Micheli. Networks on Chips : A new SoC paradigm. *Computer*, 35 :70–78, 2002.
5. Jose Antonio Casas, Juan Manuel Moreno, Jordi Madrenas, et Joan Cabestany. A novel hardware architecture for self-adaptive systems. In *AHS '07 : Proceedings of the Second NASA/ESA Conference on Adaptive Hardware and Systems*, pages 592–599, Washington, DC, USA, 2007. IEEE Computer Society.
6. Robit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, et Ramesh Menon. *Parallel programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.
7. Jérémie Crenne, Pierre Bomel, Guy Gogniat, et Jean-Philippe Diguet. End-to-end bitstreams repository hierarchy for FPGA partially reconfigurable systems. In Guy Gogniat, Dragomir Milojevic, Adam Morawiec, et Ahmet Erdogan, editors, *Algorithm-Architecture Matching for Signal and Image Processing*, volume 73 of *Lecture Notes in Electrical Engineering*, pages 171–194. Springer, 2011.
8. François Duhem, Fabrice Muller, et Philippe Lorenzini. FaRM : Fast reconfiguration manager for reducing reconfiguration time overhead on FPGA. In *7th International Symposium on Applied Reconfigurable Computing (ARC 2011)*, Belfast, United Kingdom, March 2011.
9. Esam El-Araby, Ivan Gonzalez, et Tarek El-Ghazawi. Exploiting partial runtime reconfiguration for High-Performance Reconfigurable Computing. *ACM Trans. Reconfigurable Technol. Syst.*, 1 :21 :1–21 :23, January 2009.
10. Tarek El-Ghazawi, Esam El-Araby, Miaoqing Huang, Kris Gaj, Volodymyr Kindratenko, et Duncan Buell. The promise of High-Performance Reconfigurable Computing. *Computer*, 41 :69–76, February 2008.
11. Milad El Khodary, Jean-Philippe Diguet, Guy Gogniat, Fabrice Muller, et Michel Auguin. On simulating operating environment decisions in a sane network. In *2nd ÆTHER - MORPHEUS Workshop-Autumn School From Reconfigurable to Self - Adaptive Computing (IST AMWAS 08)*, Lugano, Switzerland, 2008.
12. Clément Foucher, Fabrice Muller, et Alain Giulieri. Exploring FPGAs capability to host a HPC design. In *28th Norchip Conference (Norchip 2010)*, pages 1–4, Tampere Finlande, November 2010.
13. William Gropp, Ewing Lusk, Nathan Doss, et Anthony Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Comput.*, 22 :789–828, September 1996.
14. H. Peter Hofstee. Power efficient processor architecture and the Cell processor. In *HPCA '05 : Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 258–262, Washington, DC, USA, 2005. IEEE Computer Society.
15. Intel Labs., http ://techresearch.intel.com/spaw2/uploads/files/SCC_EAS.pdf. *SCC External Architecture Specification (EAS)*, March 2010.
16. R. A. Iushchenko. Measuring the performance of parallel computers with distributed memory. *Cybernetics and Sys. Anal.*, 45 :941–951, November 2009.
17. Philippe Marquet, Simon Duquennoy, Sébastien Le Beux, Samy Meftali, et Jean-Luc Dekeyser. Massively parallel processing on a chip. In *Proceedings of the 4th international conference on Computing frontiers*, CF '07, pages 277–286, New York, NY, USA, 2007. ACM.
18. Farooq Muhammad, Fabrice Muller, et Michel Auguin. Dynamic and self adaptive resource management : æther operating environment. In *3rd IEEE Int. Conference on Emerging Technologies*, Islamabad, november 2007.
19. P. David Stotts et William Pugh. Parallel finite automata for modeling concurrent software systems. *J. Syst. Softw.*, 27 :27–43, October 1994.
20. Erich Strohmaier, Jack J. Dongarra, Hans W. Meuer, et Horst D. Simon. The marketplace of high-performance computing. *Parallel Comput.*, 25 :1517–1544, December 1999.
21. Xian-He Sun et Yong Chen. Reevaluating amdahl's law in the multicore era. *J. Parallel Distrib. Comput.*, 70 :183–188, February 2010.
22. Dong Hyuk Woo et Hsien-Hsin S. Lee. Extending Amdahl's law for energy-efficient computing in the many-core era. *Computer*, 41 :24–31, 2008.
23. Xilinx, Inc., http ://www.xilinx.com/support/documentation/user_guides/ug191.pdf. *Virtex-5 FPGA Configuration User Guide*, August 2010.